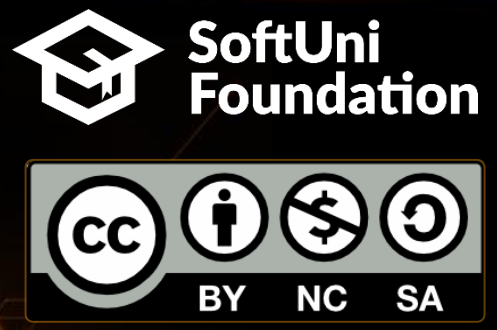
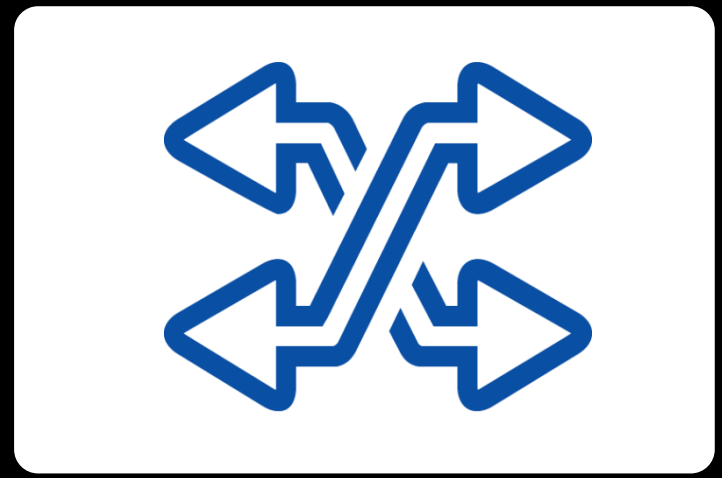
Interface Segregation / Dependency Inversion

Building Solid Code





SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Questions

SoftUni Foundation

sli.do

#JavaFundamentals

SoftUni Foundation

Dependency Inversion Principle

Would you solder a lamp directly
to the electrical wiring in a wall?

Dependency Inversion

Flip Dependencies

Dependency Inversion Principle

SoftUni Foundation

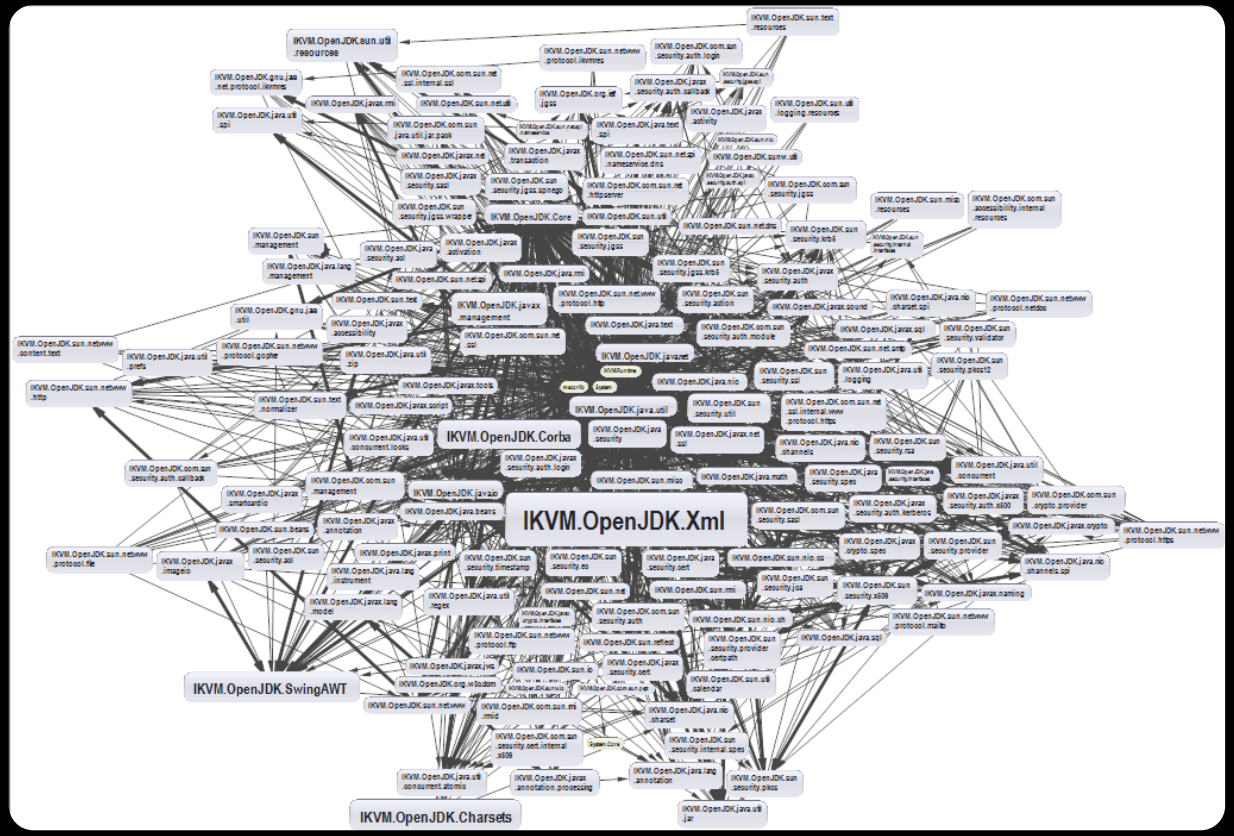
> *"Dependency Inversion Principle says that high-level modules should not depend on low-level modules. Both should depend on abstractions."*
>
> *"Abstractions should not depend on details. Details should depend on abstractions."*
>
> Agile Principles, Patterns, and Practices in C#

- Goal: **decoupling between modules** through abstractions

Dependencies and Coupling

SoftUni Foundation

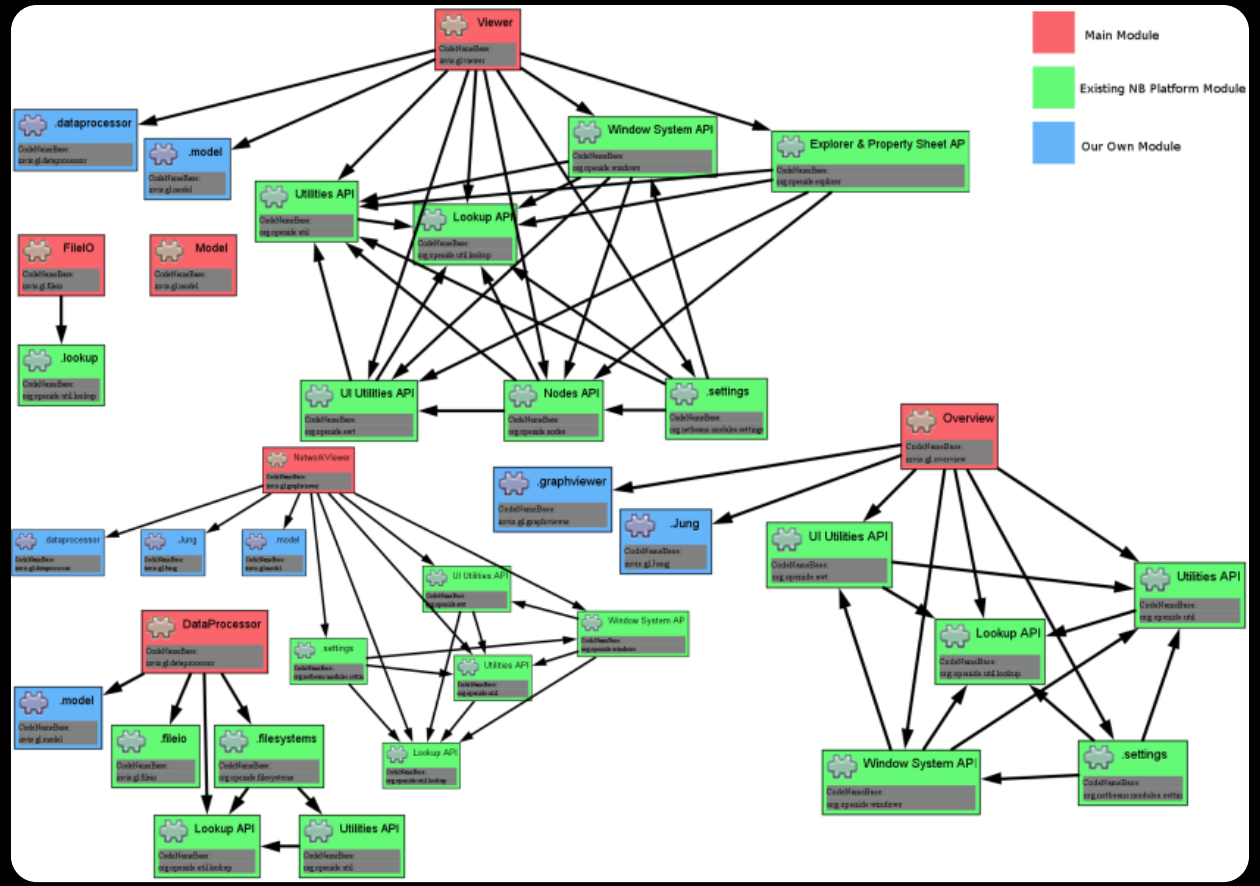
- What happens when modules **depend directly** on **other modules**



Dependencies and Coupling (2)

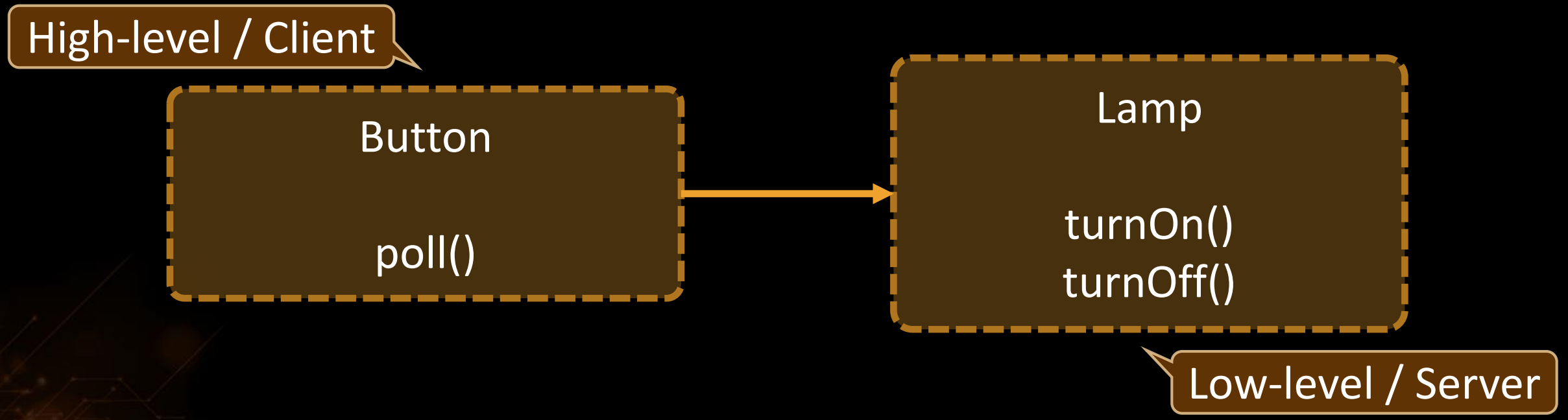
SoftUni Foundation

- The goal is to **depend on abstractions**



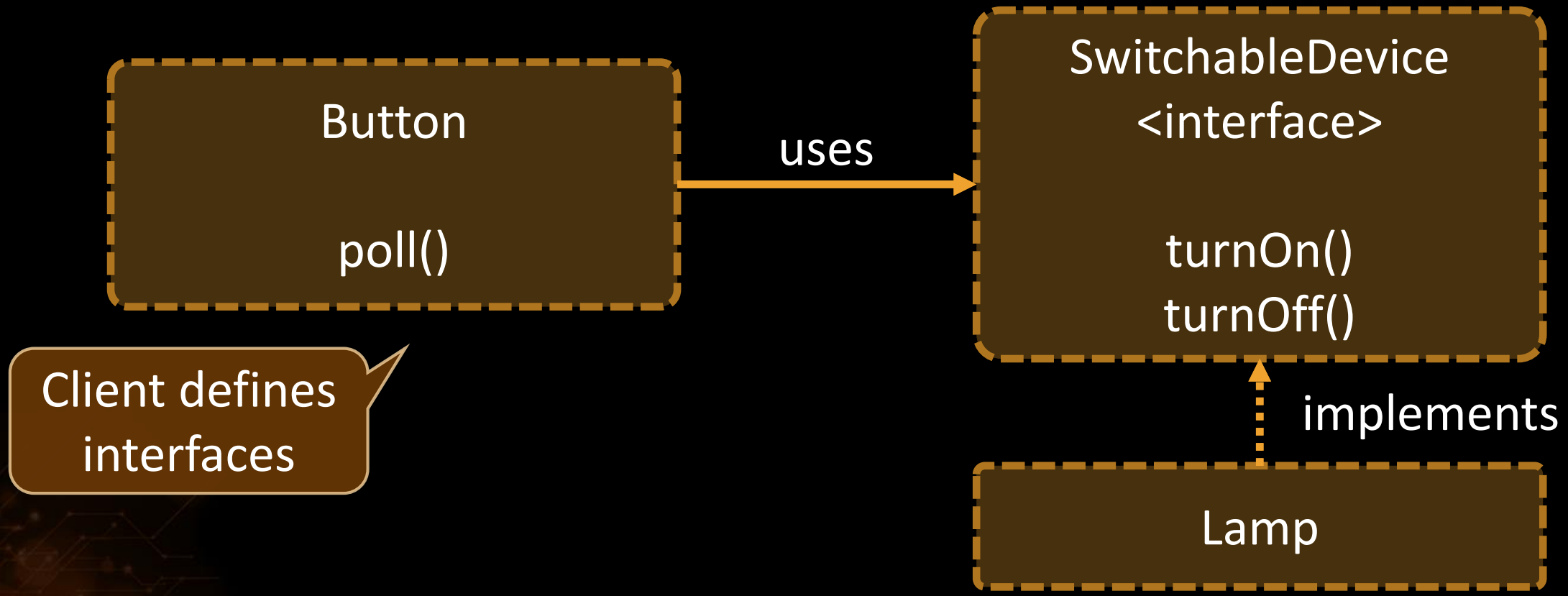
The Problem

- Button → Lamp Example – **Robert Martin**
- Button **depends on** Lamp



Dependency Inversion Solution

- Find the abstraction independent of details



Dependency Examples

SoftUni Foundation

- A dependency is any external component / system:
  - Framework
  - Third party library
  - Database
  - File system
  - Email
  - Web service
  - System resource (e.g. clock)
  - Configuration
  - The `new` keyword
  - Static method
  - Global function
  - Random generator
  - System.in / System.out

How to DIP?

SoftUni Foundation

- **Constructor injection**
  - Dependencies are passed through **constructors**
    - **Pros**
      - Classes **self-documenting** requirements
      - Works well without container
      - Always **valid state**
    - **Cons**
      - Many parameters
      - Some methods may not need everything

Constructor Injection – Example

SoftUni Foundation

```
public class Copy {
  private Reader reader;
  private Writer writer;
  public Copy(Reader reader, Writer writer) {
        this.reader = reader;
        this.writer = writer;
  }
  public void copyAll() {}
}
```

How to DIP? (2)

SoftUni Foundation

- **Setter Injection**
  - Dependencies are passed through **setters**
    - **Pros**
      - Can be changed anytime
      - Very **flexible**
    - **Cons**
      - Possible **invalid state** of the object
      - Less intuitive

Setter Injection – Example

SoftUni Foundation

```
public class Copy {
  private Reader reader;
  private Writer writer;
  public void setReader(Reader reader) {}
  public void setWriter(Writer writer) {}
  public void copyAll() {}
}
```

How to DIP? (3)

SoftUni Foundation

- **Parameter injection**
  - Dependencies are passed through **method parameters**
  - **Pros**
    - No change in rest of the class
    - Very flexible
  - **Cons**
    - Many parameters
    - Breaks the method signature

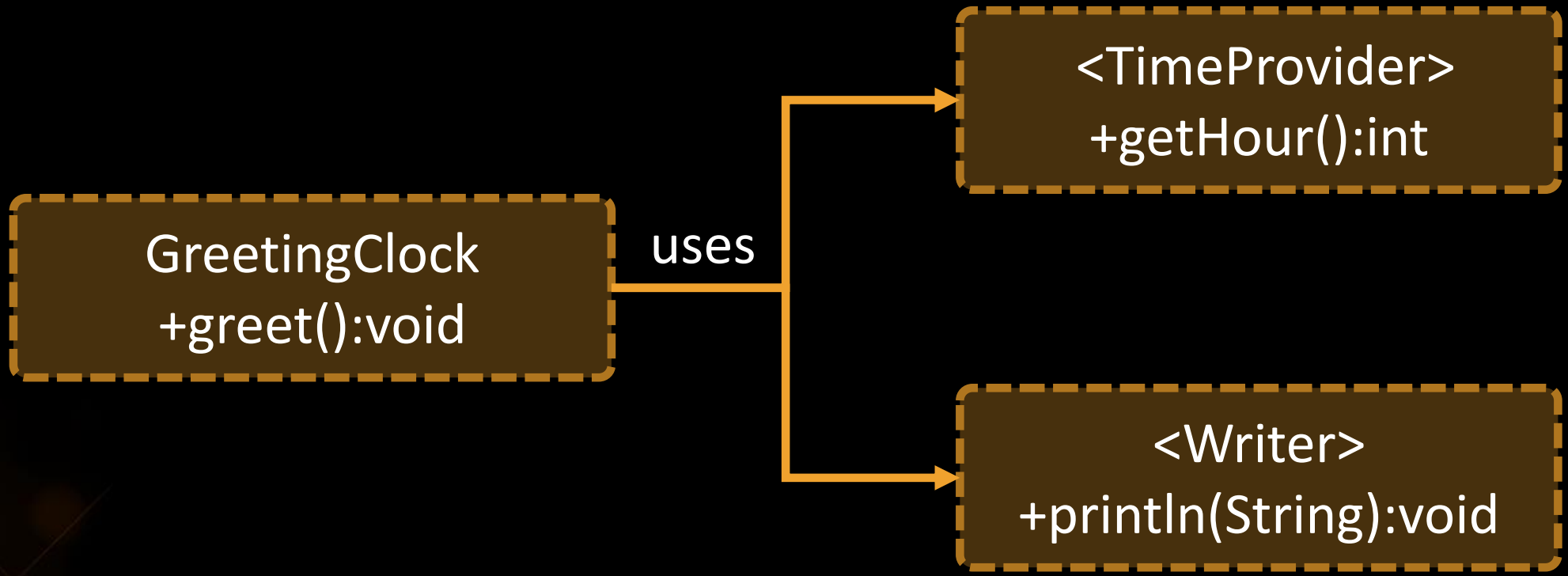
```
public class Copy {
  public copyAll(Reader reader, Writer writer) {}
}
```

Problem: System Resources

- You are given a **`GreetingClock`**
  - if hour < 12, prints **"Good morning…"**
  - if hour < 18, prints **"Good afternoon…"**
  - else prints **"Good evening…"**
- Refactor so it conforms to DIP
- * Introduce **Strategy Design Pattern**

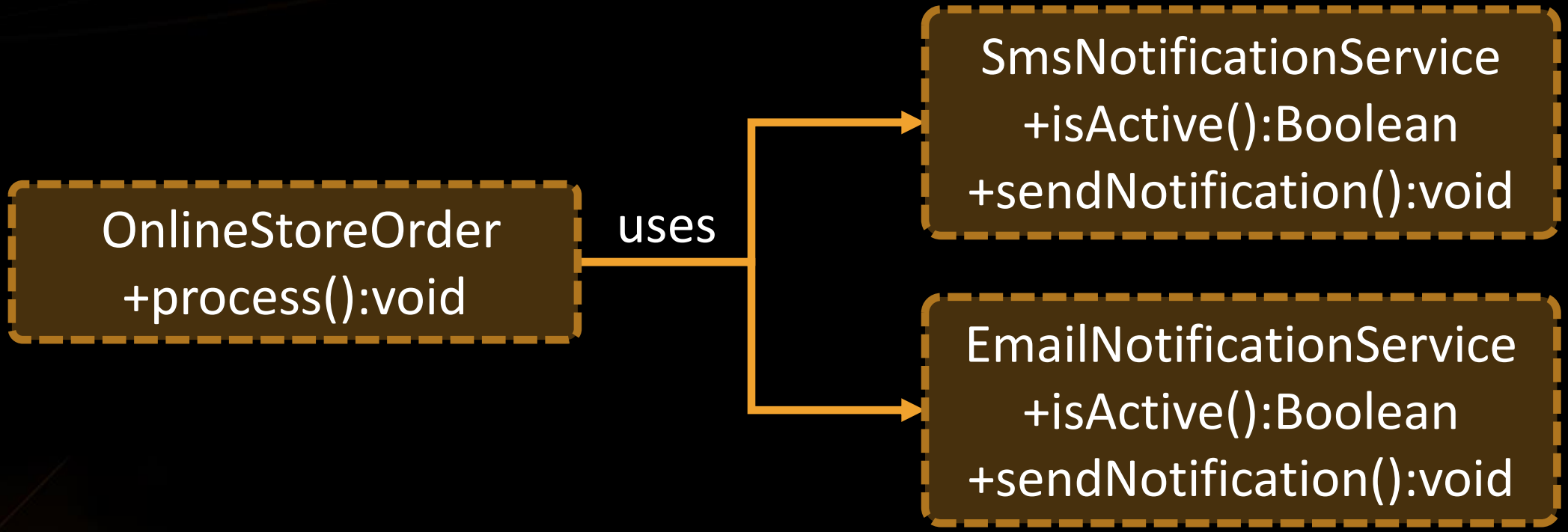
Solution: System Resources

- Inject interfaces `TimeProvider` and `Writer`



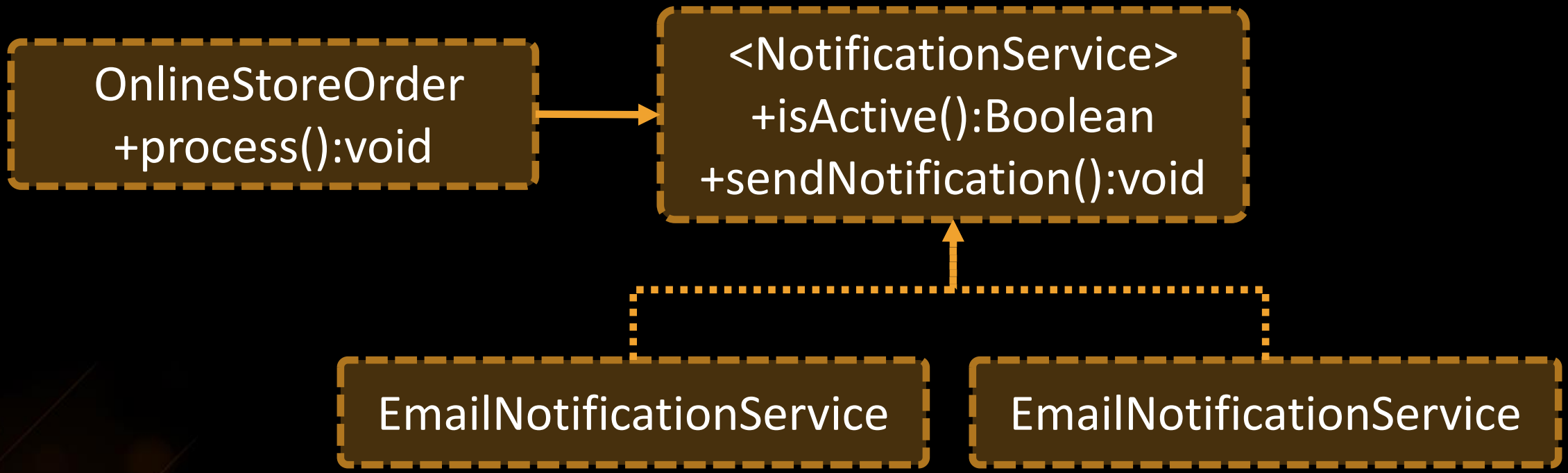
Problem: Services

- You are given some classes



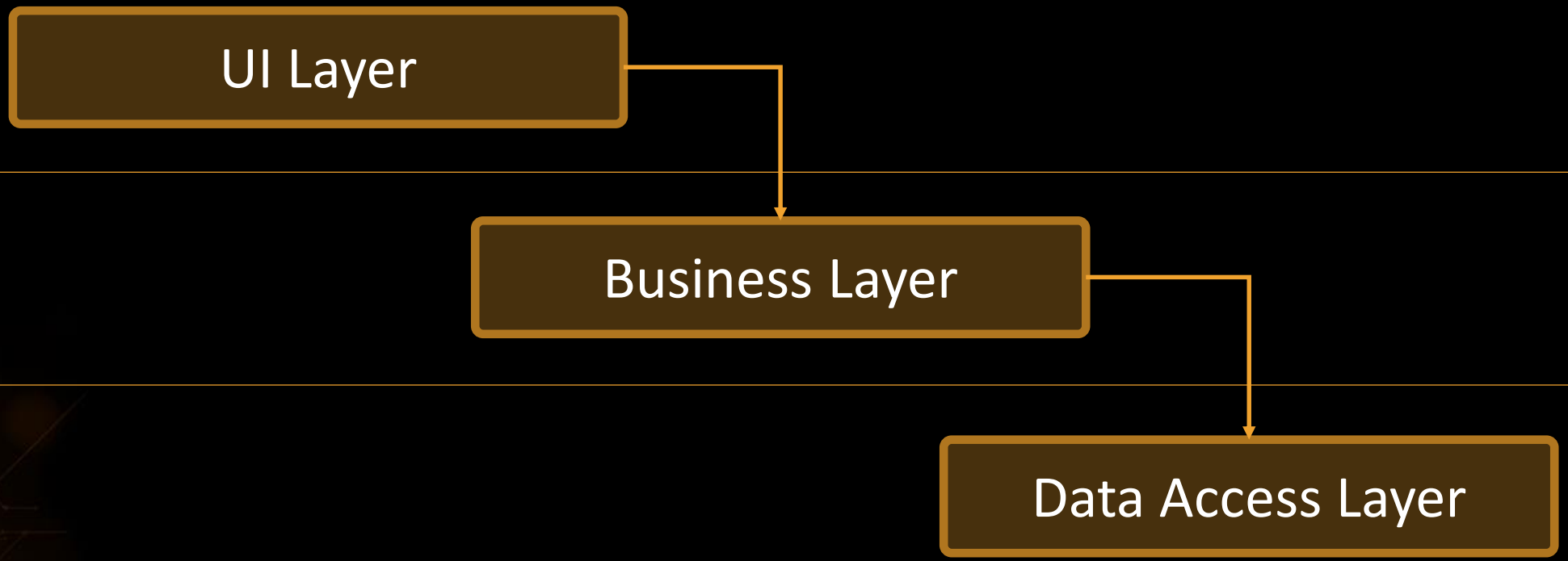
- Follow **DIP** to **invert dependencies**
- *Introduce **Composite Design Pattern**

Solution: Services



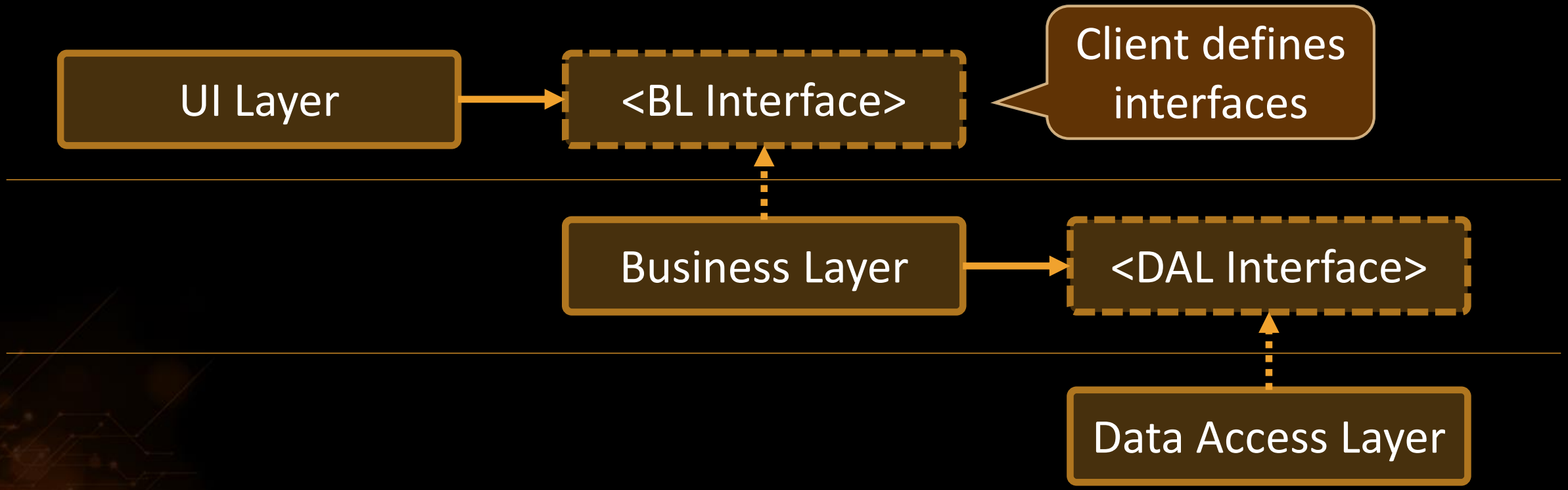
Layering

- Traditional programming
  - **High-level** modules use **low-level** modules



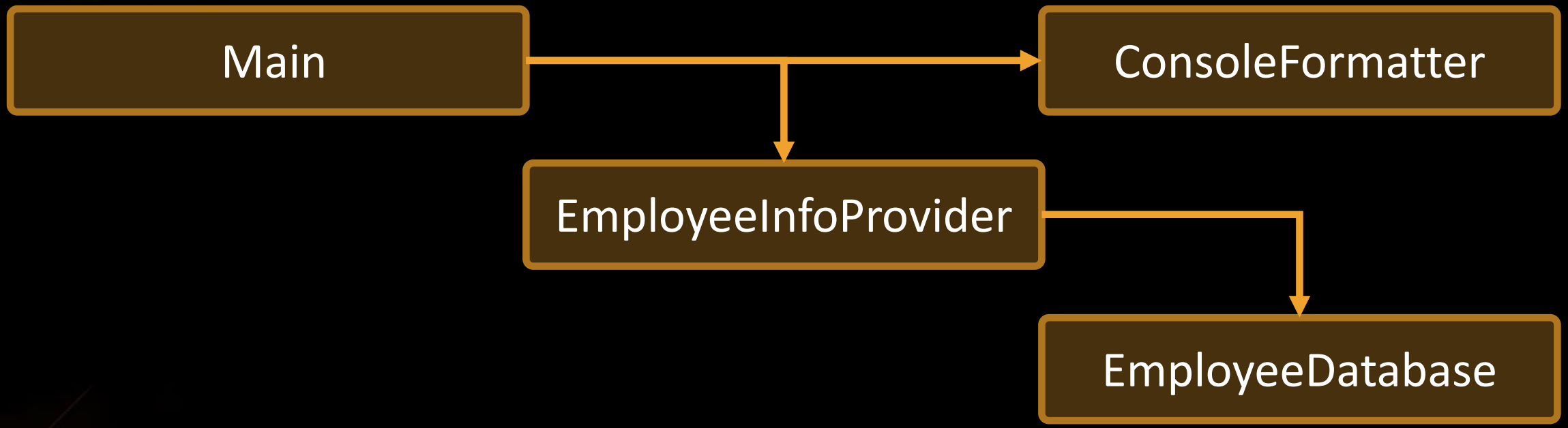
Layering (2)

- Dependency Inversion Layering
  - **High** and **low-level** modules **depend on abstractions**



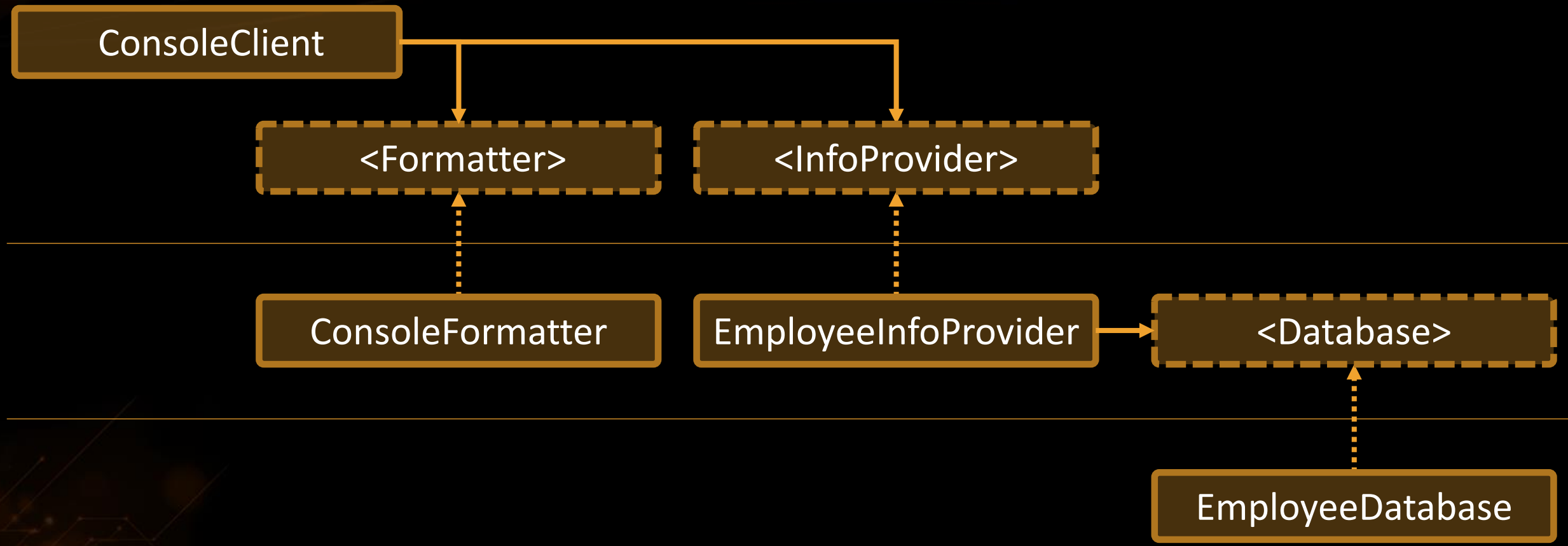
Problem: Employee Info

- You are given some classes



- Refactor the code so that it conforms to DIP

Solution: Employee Info



SoftUni Foundation

Dependency Inversion Principle

Live Exercises in Class (Lab)

SoftUni Foundation

Interface Segregation

Clients Require Cohesive Interfaces

ISP – Interface Segregation Principle

SoftUni Foundation

> *"Clients should not be forced to depend on methods they do not use."*
>
> Agile Principles, Patterns, and Practices in C#

- Segregate interfaces
  - Prefer **small**, **cohesive** interfaces
  - Divide "**fat**" interfaces into "**role**" interfaces

Fat Interfaces

SoftUni Foundation

- Classes whose interfaces are not cohesive have "fat" interfaces

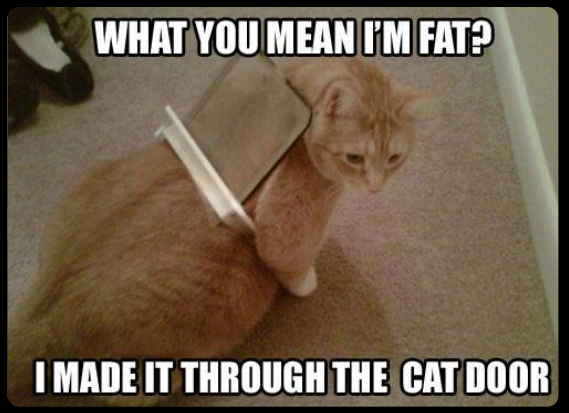
```
public interface Worker {
  void work();
  void sleep();
}
```

Class **Employee** is OK

```
public class Robot implements Worker {
  void work() {}
  void sleep() {
    throw new UnsupportedOperationException();
  }
}
```

"Fat" Interfaces

- Having "**fat**" interfaces:
  - Classes have methods they do not use
  - Increased **coupling**
  - Reduced flexibility
  - Reduced maintainability



How to ISP?

SoftUni Foundation

- Solutions to broken ISP
  - **Small** interfaces
  - **Cohesive** interfaces
  - Let the **client define interfaces** – **"role"** interfaces

Cohesive Interfaces

SoftUni Foundation

- Small and Cohesive "**Role**" Interfaces

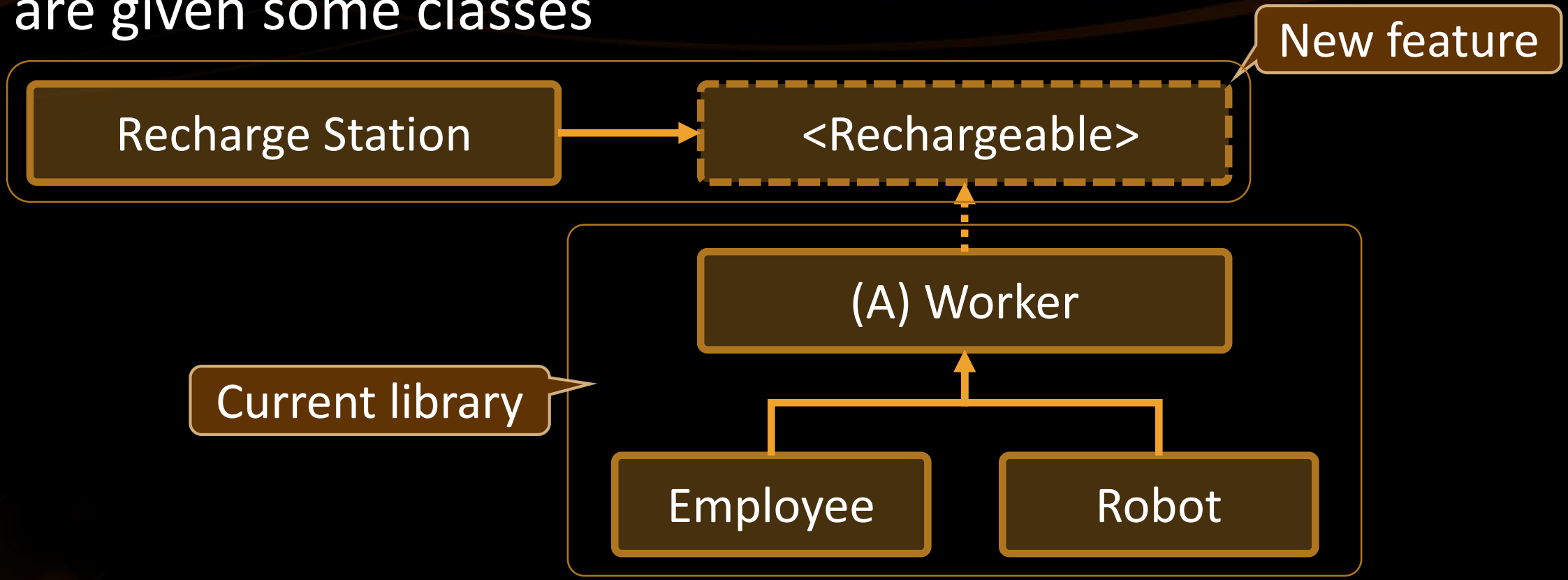
```
public interface Worker {
  void work();
}
```

```
public interface Sleeper {
  void sleep();
}
```

```
public class Robot implements Worker {
  void work() {
    // Do some work…
  }
}
```

Problem: Recharge

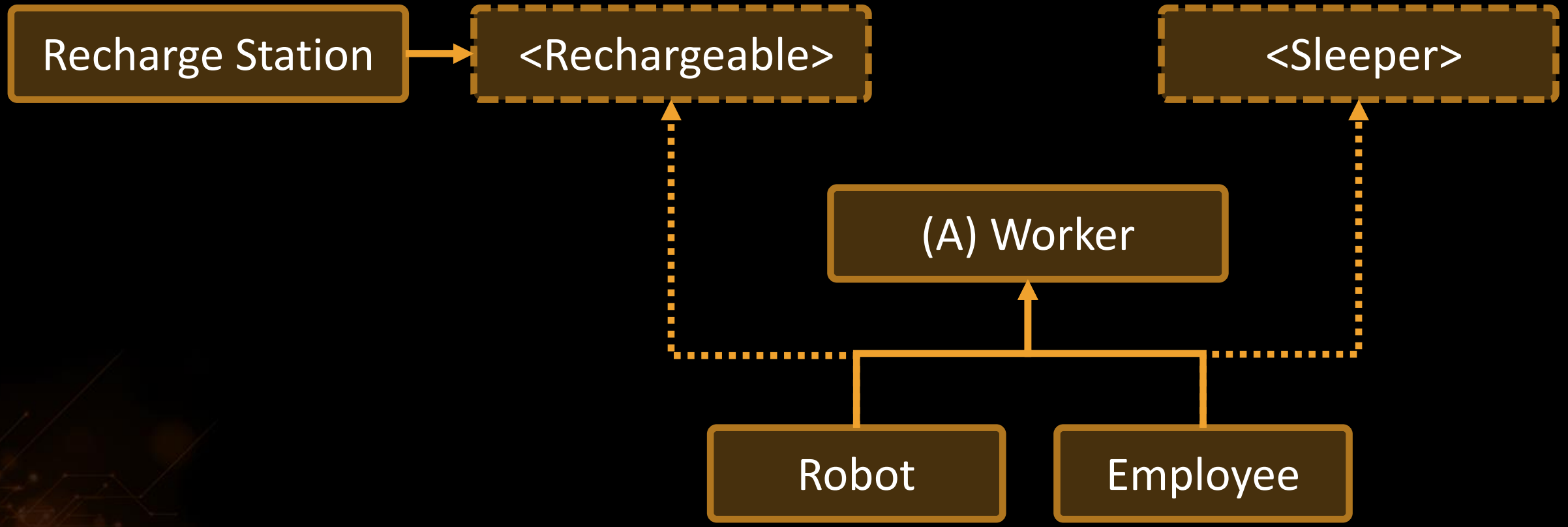
- You are given some classes



- Refactor the code so that it **conforms to ISP**
- * Consider the case that you **don't own the library**

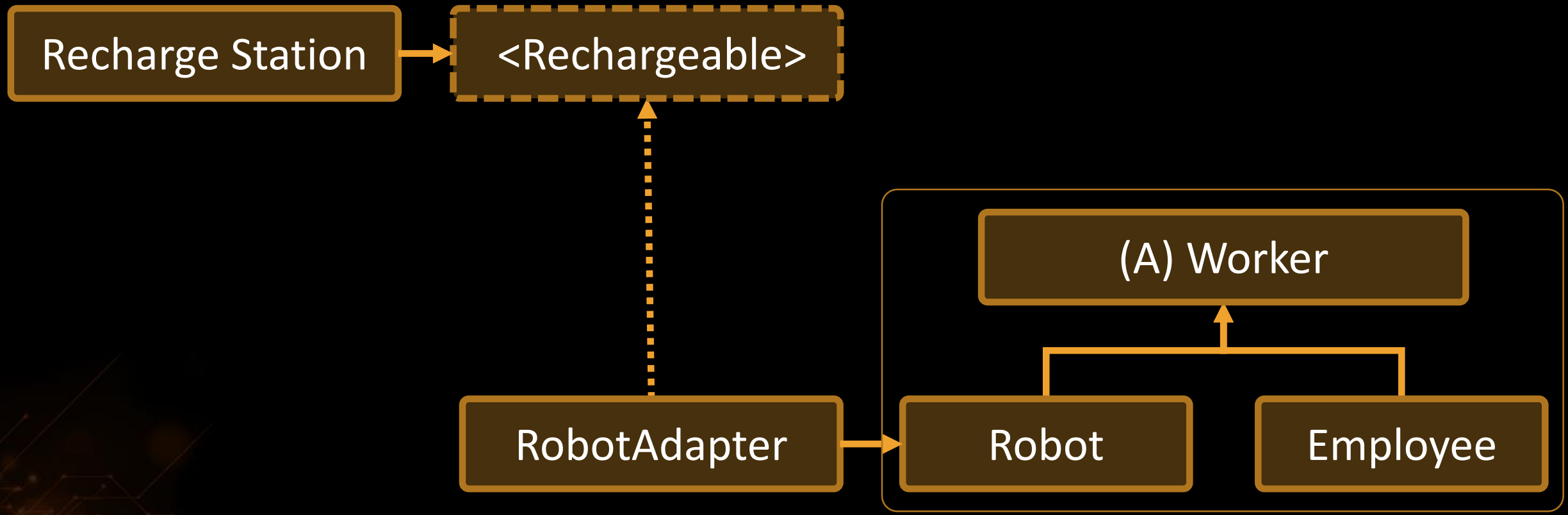
Solution: Rechargeable

- Multiple Inheritance (If you own the library)



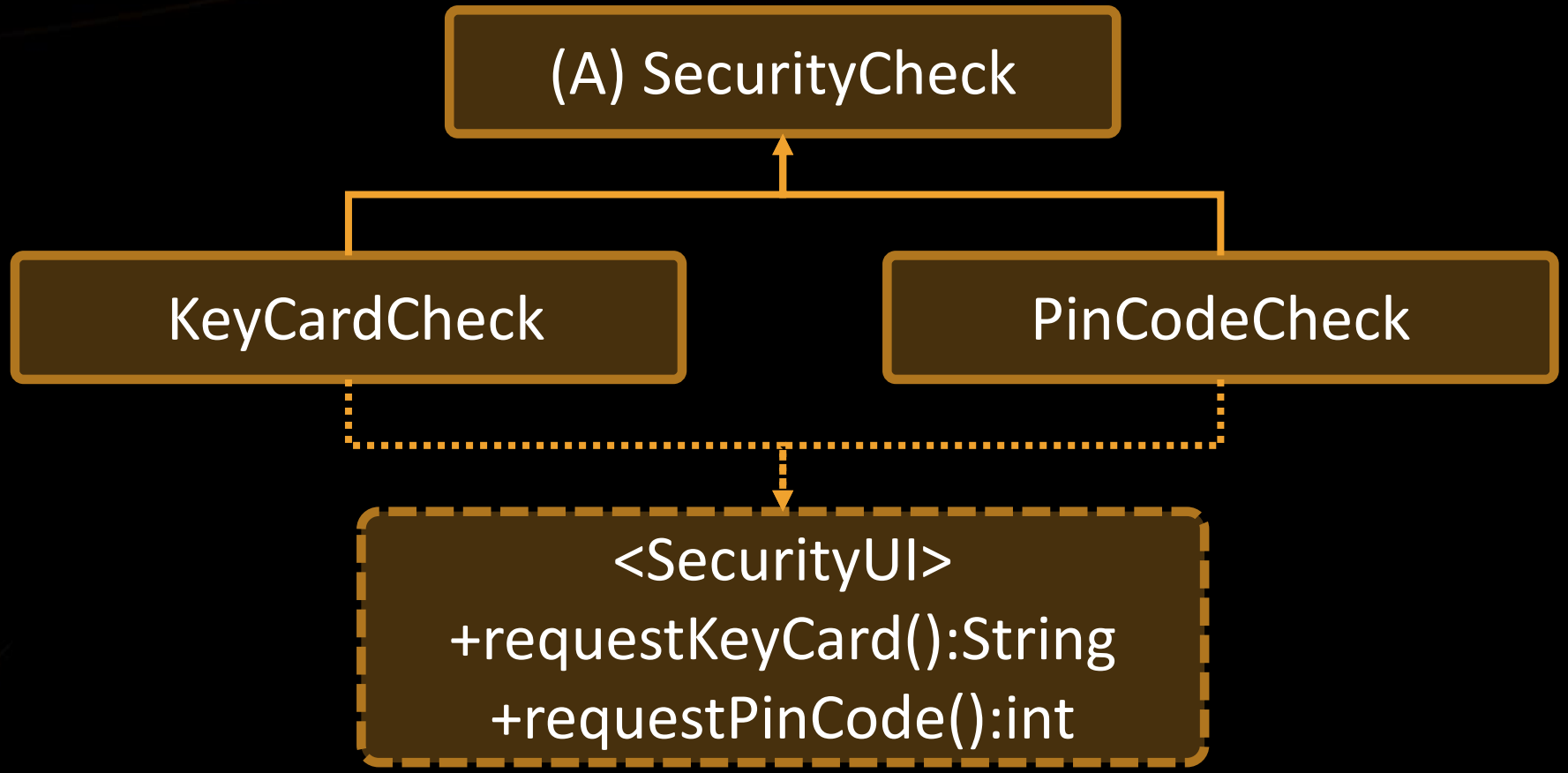
Solution: Rechargeable (2)

- Adapter Pattern (If you don't own the library)



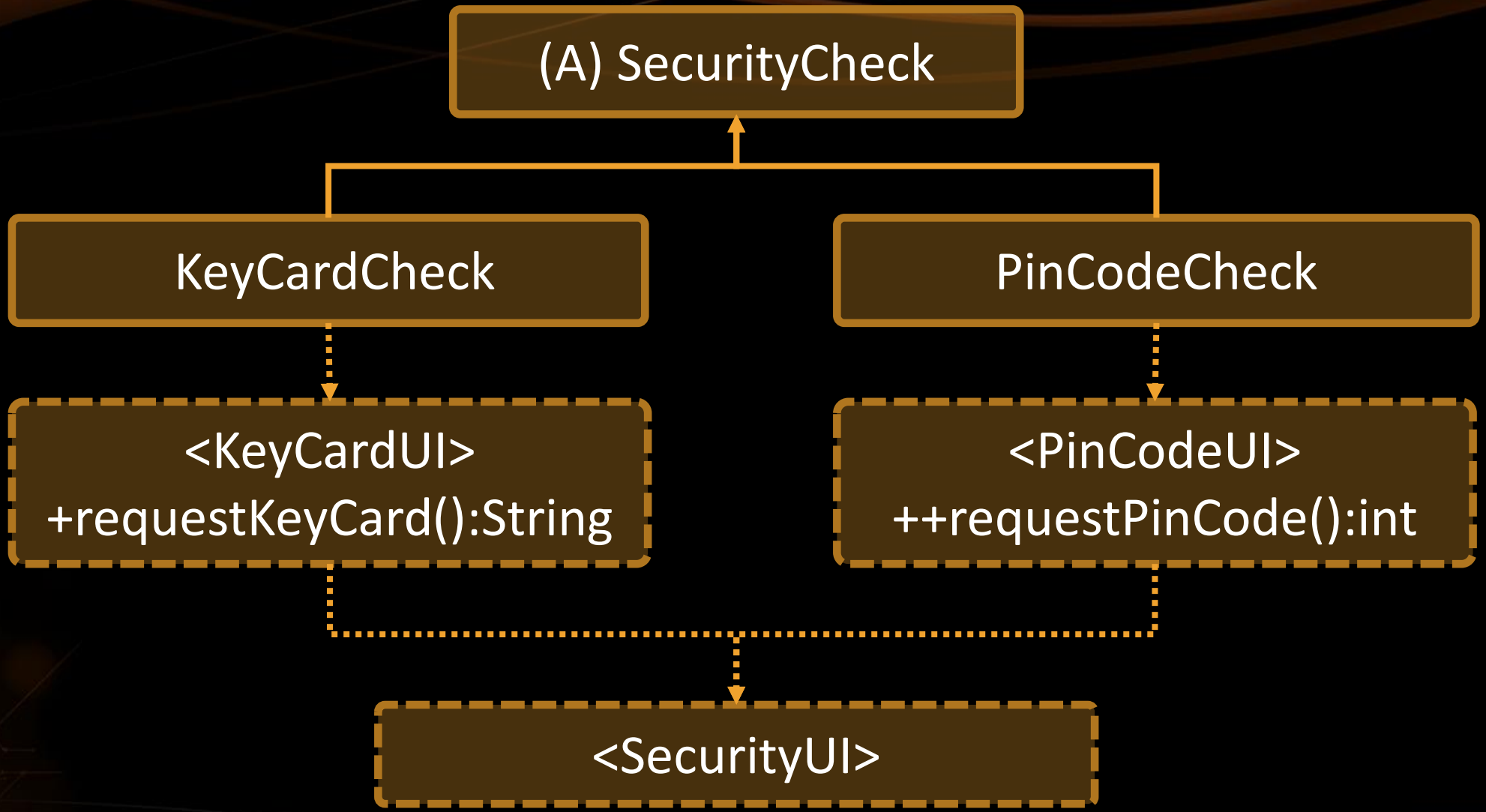
Problem: Security Door

- You are given some classes



- Refactor the code so that it **conforms to ISP**

Solution: Security Door



Summary

- Use **dependency injection** and **abstractions**
- **High** and **low-level** modules should **depend on abstractions**
- Abstractions should **not depend on details**
- Let the **client define the interfaces**
- Prefer "**role**" interfaces

Interface Segregation / Dependency Inversion

SoftUni Foundation

Questions?

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

SoftUni Foundation

FORUM

License

SoftUni Foundation

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Fundamentals of Computer Programming with Java" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "C# Part I" course by Telerik Academy under CC-BY-NC-SA license
  - "C# Part II" course by Telerik Academy under CC-BY-NC-SA license